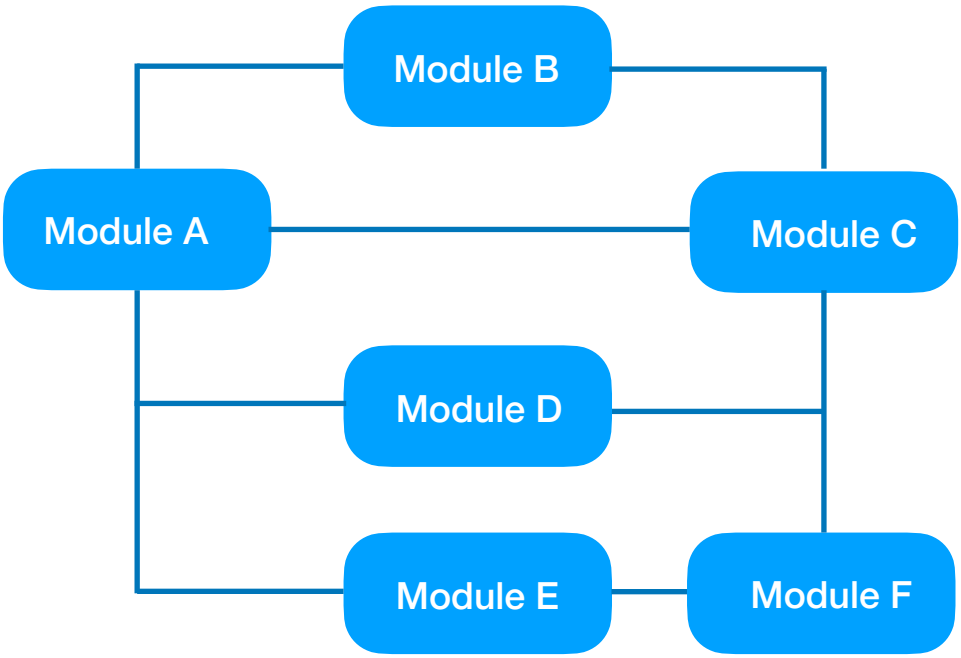
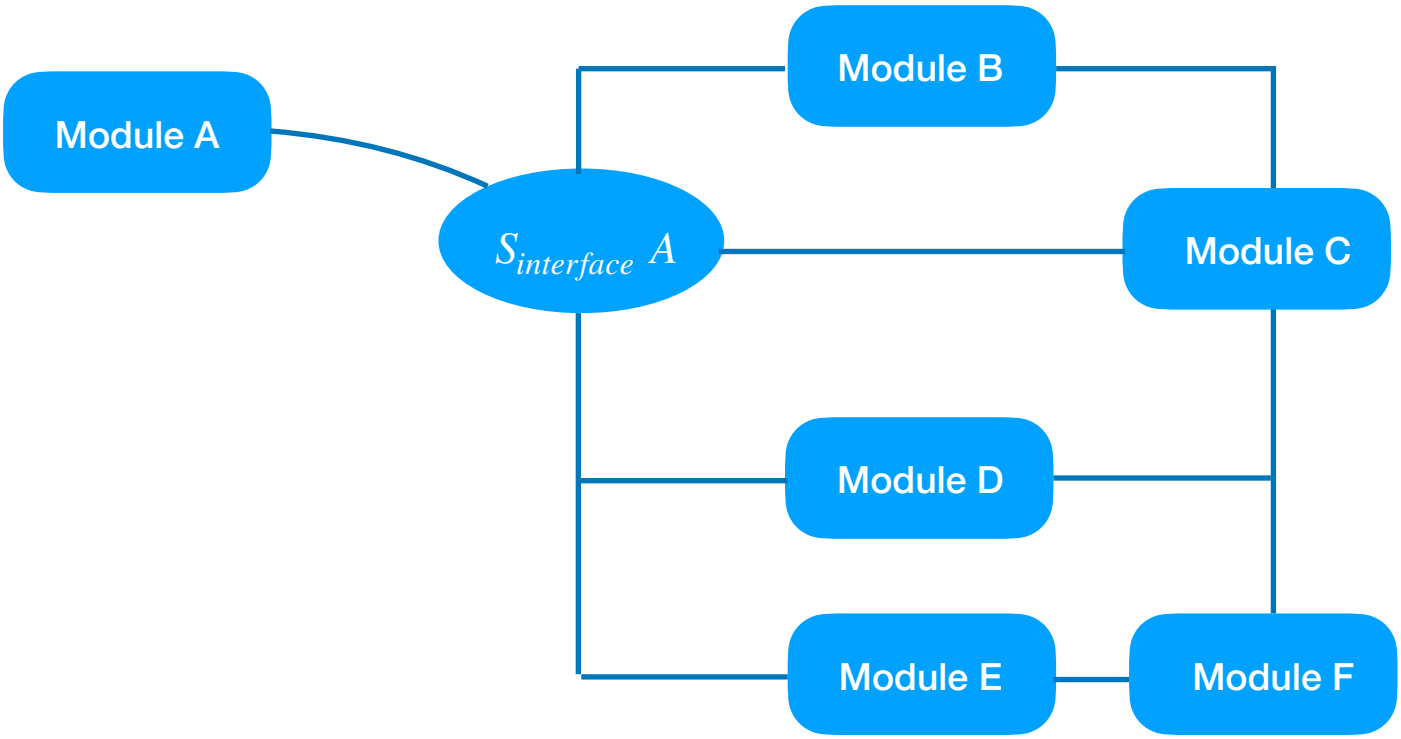
1. Покажете с подходяща структура, подход за увеличаване на възможностите за промяна на модул A (modifiability) в следната диаграма. Обосновете отговора си. (2 т.)





Тъй като модул A е свързан с всички останали модули от системата, с изключение на F, то промяната му ще доведе до метода на вълната и ще се наложи да направим и промени в модулите B, C, D и E, за да ги синхронизираме отново с вече променения модул A. За да предотвратим нежелания ефект, ще декомпозираме отговорността на модула A като използваме тактиката „скриване на информацията“. Публичната функционалност на A ще е достъпна посредством специално дефинирани интерфейси.

2. Обяснете смисъла на групата тактики за отлагане на свързването - за удовлетворяване на какви изисквания се прилагат и какво представляват. (1 т.)

Това са тактики за изменяемост и промени по време на експлоатация на системата. Тук промените не се правят от разработчиците, а от актьори, които са от страна на клиента - краини потребители, системни администратори. Очевидно тези промени нямат нищо общо с промяната на броя модули, но за да бъде възможна направата им е наложително да се инвестира в допълнителна инфраструктура, която в случая наричаме именно „отлагане на свързването“. Пачовете, които често инсталираме сами дори и без да разберем са идеален пример за „отлагане на свързването“ или когато искаме да гледаме някаква мултимедия, често ни се налага да добавим някакви нови функционалности и те да останат при нас под формата на инсталирани plugin-и. Друг интересен пример е средата за разработка eclipse. Тя е направена с едно ядро, но има модул plugin, които когато се стартира - проверява всички xml файлове за да разбере кои са допълнителните plugin-и, които са инсталирани. Това е сравнително лесно за реализация но изисква допълнително да се помисли включването на тази инфраструктура към архитектурата.

3. Опишиете подходящ според вас подход за справяне с атака към сигурността от типа Denial of Service. (3 т.)

Оточняване на проблема: Denial of Service (DoS) атаката е атака за отказ на услуга и опит да се направи дадена система недостъпна за реалните ѝ потребители. Този вид атака се осъществява като злонамерен участник в кибер пространството (направи опит и успее да) „наводни“ мрежовия сървър с трафик докато той не може да реагира повече и просто се срине, предотвратявайки възможността за достъп на законните потребители. Злонамереният участник може да изчерпа или пропускателния достъп на системата или изчислителните ѝ ресурси на сървъра (памет, процесорно време и т.н.). И в двата случая системата няма да може да бъде достъпвана от реалните потребители.

Подходи за справяне с атака към сигурността от типа Denial of Service:
- Въвеждане на автентикация на потребителите. Например ако само регистриран потребител може да използва услугите на системата, то при регистрация ще трябва да разпознава в кое от квадратчетата на подадена снимка има част от жираф.
- Въвеждане на ограничение на експлоатацията на местата където са уязвими на атака. Например, ако сървъра може да обработва по 1000 заявки за някакво изчиление, то всяко следващо ще чака. Но това пак не е решение за DoS SYN Flood типа досовска атака, където нападателят оставя заявките без отговор и така те остават незавършени и заемат мрежови ресурси. За това може да се въведе и времеви лимит за изпълнението на всяка заявка, след което тя да се прекратява автоматично.

Някой по-сериозни мерки, които може да предложим и комбинираме:
- Наблюдаване на трафика с цел установяване възможно най-РАНО на DoS атаката. За целта трябва да знаем какво означава ниско, средно и високо натоварване на системата, като отчитаме сезонност, маркетингови кампании и т.н. . Така ще минимизираме загубите.
- Ще ограничим достъпа с допълнителен слой защита чрез средства за ограничаване на по-нататъшен достъп като DMZ или firewall. Може при необходимост дори да използваме двойни защитни стени (периметърна и вътрешна)
- Ако допуснем, че приложението генерира сериозен интерес и трафик от различни части на света, то ще съхраняваме данните на няколко сървъра по целия сват. Това го правят именно CDN мрежите. Те услужват данните към потребителите от сървър, който е най-близо до тях с цел по-бърза връзка, НО това автоматично прави и системата по-малко уязвима на такива атаки, защото има много други сървъри, които функционират.

- Симулиране на DoS атаки срещу собствената си система, за да изпробваме текущите методи за превенция.
- Създаване на готов план за управление на щетитие и отговор след атаката. Класифициране на атаките (неудобни / вредни / пагубни и т.н.) и ще посочим роли в екипа, които ще отговарят за реакцията след всеки един от сценариите.
- Използване на готови услуги StackPath Edge Services за разпознаване и блокиране на DoS атаките в реално време.

4. Какъв архитектурен стил (или комбинация от стилове) ще използвате за да повишите производителността на системата, в която се обменят голямо количество данни? Защо? Обяснете смисъла на избрания стил. (3 т.)

За системата, в която се обменят голямо количество данни е високоефективно да се използва архитектурния стил на споделените данни (shared-data style). В този стил, именно голямото количество данни, които се споделят могат да играят ролята на връзка между отделните компоненти на системата, които ги консумират. Така автоматично обменът на големите количества данни вече няма да представлява пречка за производителността. По отношение на това дали споделените данни информират компонентите си консуматори, при настъпване на интересни данни или не, тези стилове се делят на Blackboard (черна дъска) и Repository (хранилище). *Забележка: под „интересни“ данни ще визираме някакъв предварително дефиниран тип данни.* В първия случай споделените данни могат да се разглеждат като активен агент, докато във втория - като пасивен такъв. В съвременните системи тези разграничения са замъглени, тъй като много системи за управление на база данни, които първоначално са били хранилища, осигуряват задействащ механизъм, който ги превръща с черни дъски.

Този стил има значителни предимства, като освен, че унищожава проблемите с производителността при обмена на голямо количество данни, той позволява на системата да бъде лесно мащабируема (scalability), като и предоставя възможност за добавяне на компоненти, които лесно могат да се свържат със споделените данни. При разработка на такъв стил, всички компоненти на системата могат да се разработват паралелно, което ще спести доста време. Предоставя възможност за централизирано управление на данните и следователно и по-добри условия за архивиране, сигурност и т.н. . Също така компонентите не са зависими от производителя на данни.

Естествено, както при всеки един архитектурен стил, за цената на тези предимства, трябва да заплатим с някои недостатъци, като например, че този стил е труден за приложение в разпределени системи. Не е удобен за преизползване поради зависимостта от споделените данни. В случай на много клиенти/потребители може да стане „тясно“ и да се получи забавяне/ изчакване. Промените във форматът на съхранение на данни засягат всички модули, например, ако структурата на данните е променена, тогава всички модули трябва да бъдат модифицирани.